Variable Neighborhood Search

26

Pierre Hansen and Nenad Mladenović

Contents

Abstract

Variable neighborhood search (VNS) is a metaheuristic for solving combinatorial and global optimization problems. Its basic idea is systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. In this chapter we present the basic schemes of variable neighborhood search and some of its extensions. We next present four families of applications of VNS in which it has proved to be very successful: (i) finding feasible solutions to large mixed-integer linear programs, by hybridization of VNS and local branching, (ii) finding good feasible solutions to continuous nonlinear programs, (iii) finding programs in automatic fashion

P. Hansen (✉)
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada
e-mail: pierreh@crt.umontreal.ca

N. Mladenović
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

LAMIH, University of Valenciennes, Famars, France
e-mail: nenadmladenovic12@gmail.com

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_19

(artificial intelligence field) by building *variable neighborhood programming* methodology, and (iv) exploring graph theory in order to find conjectures, refutations, and proofs or ideas of proofs.

Keywords
Optimization · Metaheuristics · Artificial intelligence · Variable neighborhood search

Introduction

Optimization tools have greatly improved during the last two decades. This is due to several factors: (i) progress in mathematical programming theory and algorithmic design, (ii) rapid improvement in computer performances, and (iii) better communication of new ideas and integration of them in largely used complex softwares. Consequently, many problems long viewed as out of reach are currently solved, sometimes in very moderate computing times. This success, however, has led to address much larger instances and more difficult classes of problems. Many of these may again only be solved heuristically. Therefore thousands of papers describing, evaluating, and comparing new heuristics appear each year. Keeping abreast of such a large literature is a challenge. Metaheuristics or general frameworks for building heuristics are therefore needed in order to organize the study of heuristics. As evidenced by this handbook, there are many of them. Some desirable properties of metaheuristics [24,27,29] are listed in the concluding section of this chapter.

Variable neighborhood search (VNS) is a metaheuristic proposed by some of the present authors a dozen years ago [40]. Earlier work that motivated this approach is given in [8, 13, 18, 38]. It is based upon the idea of systematic change of neighborhood both in a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. Originally designed for approximate solution of combinatorial optimization problems, it was extended to address, nonlinear programs, and recently mixed-integer nonlinear programs. In addition VNS has been used as a tool for automated or computer-assisted graph theory. This led to the discovery of over 1500 conjectures in that field, the automated proof of more than half of them as well as the unassisted proof of about 400 of them by many mathematicians. Moreover, VNS methodology is recently applied in artificial intelligence as automatic programming technique.

Applications are rapidly increasing in number and pertain to many fields: location theory, cluster analysis, scheduling, vehicle routing, network design, lot sizing, artificial intelligence, engineering, pooling problems, biology, phylogeny, reliability, geometry, telecommunication design, etc. References are too numerous to be listed here, but many of them can be found in [25, 28, 29] and in the following special issues devoted to VNS: *IMA Journal of Management Mathematics* [37], *European Journal of Operational Research* [29], *Journal of heuristics* [47], *Computers and Operations Research* [45] and *Journal of Global Optimization* [11].

This chapter is organized as follows. In the next section we present the basic schemes of VNS, i.e., variable neighborhood descent (VND), reduced VNS (RVNS), basic VNS (BVNS), and general VNS (GVNS). In addition, to the list of basic schemes, we add skewed VNS (SVNS) and variable neighborhood decomposition search (VNDS). Three important extensions are presented in Section "Some Extensions": primal-dual VNS (PD-VNS), formulation space search (FSS), and recently developed variable formulation search (VFS). The remainder of the paper describes applications of VNS to several classes of large-scale and complex optimization problems for which it has proven to be particularly successful. Finding feasible solutions to large mixed-integer linear programs with VNS is discussed in Section "VNS for Mixed-Integer Linear Programming". Section "VNS for Continuous Global Optimization" addresses ways to apply VNS in continuous global optimization. Recent application of VNS in *automatic programming* field is discussed in section "Variable Neighborhood Programming". Applying VNS to graph theory per se (and not just to particular optimization problems defined on graphs) is discussed in Section "Discovery Science". Conclusions are drawn in Section "Conclusions". There the desirable properties of any metaheuristic are listed as well.

Basic Schemes

A deterministic optimization problem may be formulated as

$$\min\{f(x)|x \in X, X \subseteq \mathcal{S}\}, \tag{1}$$

where $\mathcal{S}, X, x$, and f, respectively, denote the *solution space* and *feasible set*, a *feasible solution*, and a real-valued *objective function*. If $\mathcal{S}$ is a finite but large set, a *combinatorial optimization* problem is defined. If $\mathcal{S} = \mathbb{R}^n$, we refer to *continuous optimization*. A solution $x^* \in X$ is *optimal* if

$$f(x^*) \leq f(x), \ \forall x \in X.$$

An *exact algorithm* for problem (1), if one exists, finds an optimal solution x^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $X = \emptyset$, or the solution is unbounded. Moreover, in practice, the time needed to do so should be finite (and not too long). For continuous optimization, it is reasonable to allow for some degree of tolerance, i.e., to stop when sufficient convergence is detected.

Let us denote with $\mathcal{N}_k$, $(k = 1, \ldots, k_{\max})$, a finite set of preselected neighborhood structures, and with $\mathcal{N}_k(x)$ the set of solutions in the kth neighborhood of x. Most local search heuristics use only one neighborhood structure, i.e., $k_{\max} = 1$. Neighborhoods $\mathcal{N}_k$ may be induced from one or more metric (or quasi-metric) functions introduced into a solution space S. An *optimal solution* x_{opt} (or global minimum) is a feasible solution where a minimum is reached. We call $x' \in X$ a *local minimum* of (1) with respect to $\mathcal{N}_k$ (w.r.t. $\mathcal{N}_k$ for short), if there is no solution

$x \in \mathcal{N}_k(x') \subseteq X$ such that $f(x) < f(x')$. Metaheuristics (based on local search procedures) try to continue the search by other means after finding the first local minimum. VNS is based on three simple facts:

Fact 1 *A local minimum w.r.t. one neighborhood structure is not necessarily so for another;*

Fact 2 *A global minimum is a local minimum w.r.t. all possible neighborhood structures;*

Fact 3 *Local minima w.r.t. one or several $\mathcal{N}_k$ are relatively close to each other, for many problems.*

This last observation, which is empirical, implies that a local optimum often provides some information about the global one. This may for instance be several variables with the same value in both. However, it is usually not known which ones are such. An organized study of the neighborhoods of this local optimum is therefore in order, until a better solution is found.

In order to solve (1) by using several neighborhoods, facts 1–3 can be used in three different ways: (i) deterministic, (ii) stochastic, and (iii) both deterministic and stochastic. We first give in Algorithm 1 the steps of the neighborhood change function that will be used later.

Algorithm 1: Neighborhood change

Function `NeighborhoodChange` (x, x', k)
1 **if** $f(x') < f(x)$ **then**
2 $\quad x \leftarrow x'$; $k \leftarrow 1$ // Make a move
else
3 $\quad k \leftarrow k + 1$ // Next neighborhood

Function `NeighborhoodChange()` compares the new value $f(x')$ with the incumbent value $f(x)$ obtained from the kth neighborhood (line 1). If an improvement is obtained, k is returned to its initial value and the new incumbent updated (line 2). Otherwise, the next neighborhood is considered (line 3).

(i) The *variable neighborhood descent* (VND) method is obtained if a change of neighborhoods is performed in a deterministic way. Its steps are presented in Algorithm 2, where neighborhoods are denoted as $N_\ell, \ell = 1, \ell_{\max}$.

Most local search heuristics use in their descents a single or sometimes two neighborhoods ($\ell_{\max} \leq 2$). Note that the final solution should be a local minimum w.r.t. all $\ell_{\max}$ neighborhoods, and thus chances to reach a global one are larger than by using a single structure. Besides this *sequential* order of neighborhood structures in VND above, one can develop a *nested* strategy. Assume, e.g., that $\ell_{\max} = 3$; then a possible nested strategy is: perform VND from Algorithm 2 for the first two neighborhoods from each point x' that belongs to the third ($x' \in N_3(x)$). Such an approach is successfully applied in [7, 25] and [5].

Algorithm 2: Steps of the basic VND

Function VND $(x, \ell_{\max})$
1 $\ell \leftarrow 1$
2 **repeat**
3 $\quad x' \leftarrow \arg\min_{y \in N_\ell(x)} f(x)$ // Find the best neighbor in $N_\ell(x)$
4 $\quad$ NeighborhoodChange (x, x', ℓ) // Change neighborhood
until $\ell = \ell_{\max}$

(ii) The *reduced VNS* (RVNS) method is obtained if random points are selected from $\mathcal{N}_k(x)$ and no descent is made. Rather, the values of these new points are compared with that of the incumbent, and updating takes place in case of improvement. We assume that a stopping condition has been chosen, among various possibilities, e.g., the maximum CPU time allowed $t_{\max}$, or the maximum number of iterations between two improvements. To simplify the description of the algorithms, we always use $t_{\max}$ below. Therefore, RVNS uses two parameters: $t_{\max}$ and $k_{\max}$. Its steps are presented in Algorithm 3.

Algorithm 3: Steps of the reduced VNS

Function RVNS$(x, k_{\max}, t_{\max})$
1 **repeat**
2 $\quad k \leftarrow 1$
3 $\quad$ **repeat**
4 $\quad\quad x' \leftarrow$ Shake(x, k)
5 $\quad\quad$ NeighborhoodChange (x, x', k)
$\quad$ **until** $k = k_{\max}$
6 $\quad t \leftarrow$ CpuTime()
until $t > t_{\max}$

With the function Shake represented in line 4, we generate a point x' at random from the kth neighborhood of x, i.e., $x' \in \mathcal{N}_k(x)$. Its steps are given in Algorithm 4, where it is assumed that points from $\mathcal{N}_k(x)$ are $\{x^1, \ldots, x^{|\mathcal{N}_k(x)|}\}$. RVNS is useful

Algorithm 4: Steps of the shaking function

Function Shake(x, x', k)
1 $w \leftarrow [1+\text{Rand}(0, 1) \times |\mathcal{N}_k(x)|]$
2 $x' \leftarrow x^w$

for very large instances for which local search is costly. It can be used as well for finding initial solutions for large problems before decomposition. It has been observed that the best value for the parameter $k_{\max}$ is often 2 or 3. In addition, a

maximum number of iterations between two improvements is usually used as the stopping condition. RVNS is akin to a Monte Carlo method, but is more systematic (see, e.g., [42] where results obtained by RVNS were 30 % better than those of the Monte Carlo method in solving a continuous min-max problem). When applied to the p-median problem, RVNS gave equally good solutions as the *fast interchange* heuristic of [56] while being 20–40 times faster [30].

(iii) The *basic VNS* (VNS) method [40] combines deterministic and stochastic changes of neighborhood. The deterministic part is represented by a local search heuristic. It consists in (a) choosing an initial solution x, (b) finding a direction of descent from x (within a neighborhood $N(x)$), and (c) moving to the minimum of $f(x)$ within $N(x)$ along that direction. If there is no direction of descent, the heuristic stops, and otherwise it is iterated. Usually the steepest descent direction, also referred to as *best improvement*, is used (`BestImprovement`). This set of rules is summarized in Algorithm 5, where we assume that an initial solution x is given. The output consists of a local minimum, also denoted by x, and its value.

Algorithm 5: Best improvement (steepest descent) heuristic

Function `BestImprovement`(x)
repeat
  $x' \leftarrow x$
  $x \leftarrow \arg\min_{y \in N(x)} f(y)$
until $(f(x) \geq f(x'))$

As *steepest descent* heuristic may be time-consuming, an alternative is to use the *first descent* heuristic. Vectors $x_i \in N(x)$ are then enumerated systematically and a move is made as soon as a direction for the descent is found. This is summarized in Algorithm 6.

Algorithm 6: First improvement (first descent) heuristic

Function `FirstImprovement`(x)
repeat
  $x' \leftarrow x$; $i \leftarrow 0$
  repeat
    $i \leftarrow i + 1$
    $x \leftarrow \arg\min\{f(x), f(x_i)\}$, $x_i \in N(x)$
  until $(f(x) < f(x_i)$ or $i = |N(x)|)$
until $(f(x) \geq f(x'))$

The stochastic phase is represented by random selection of one point from the k-th neighborhood. The steps of the BVNS are given on Algorithm 7.

Algorithm 7: Steps of the basic VNS

Function BVNS$(x, k_{\max}, t_{\max})$

1 $t \leftarrow 0$
2 **while** $t < t_{\max}$ **do**
3 $\quad k \leftarrow 1$
4 $\quad$ **repeat**
5 $\quad\quad x' \leftarrow$ Shake(x, k) // Shaking
6 $\quad\quad x'' \leftarrow$ BestImprovement(x') // Local search
7 $\quad\quad$ NeighborhoodChange(x, x'', k) //Change neighborhood
$\quad$ **until** $k = k_{\max}$
8 $\quad t \leftarrow$ CpuTime()

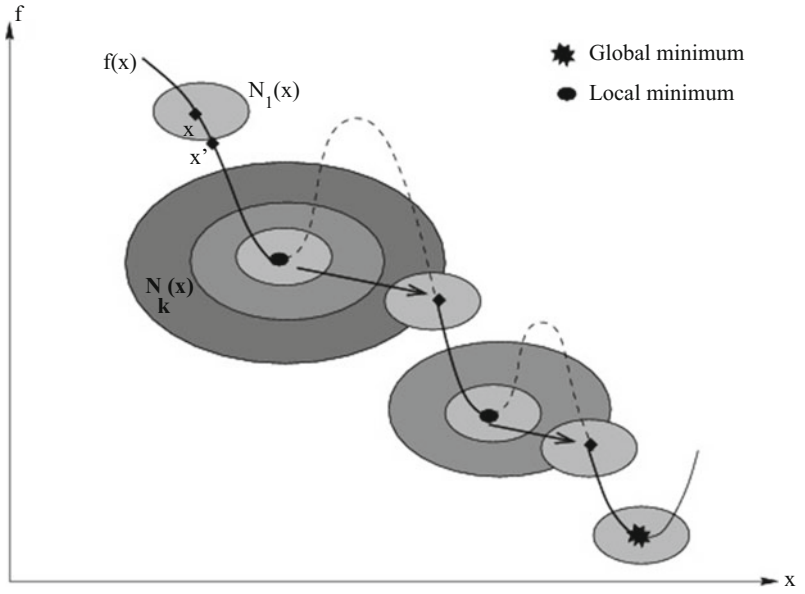

Fig. 1 Basic VNS

Often successive neighborhoods $\mathcal{N}_k$ are nested. Note that point x' is generated at random in Step 4 in order to avoid cycling, which might occur if a deterministic rule were applied. Basic VNS is also illustrated in Fig. 1.

Example. We illustrate the basic step on a minimum k-cardinality tree instance taken from [34] (see Fig. 2). The minimum k-cardinality tree problem on graph G (k-card for short) consists in finding a sub-tree of G with exactly k edges whose sum of weights is minimum.

The steps of BVNS for solving the 4-card problem are illustrated in Fig. 3. In Step 0 the objective function value, i.e., the sum of edge weights, is equal to 40;

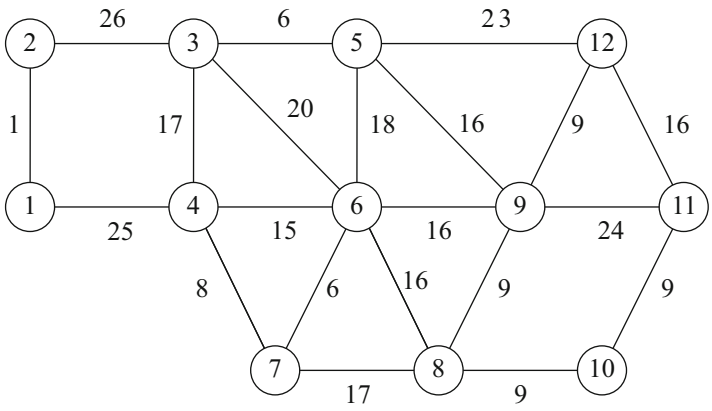

Fig. 2 4-cardinality tree problem

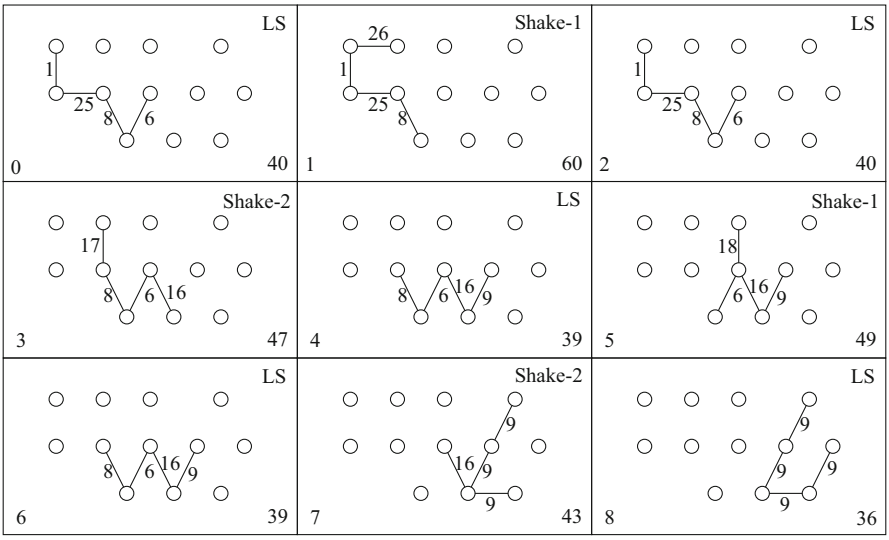

Fig. 3 Steps of the basic VNS for solving 4-card tree problem

it is indicated in the right bottom corner of the figure. That first solution is a local minimum with respect to the edge-exchange neighborhood structure (one edge in, one out). After shaking, the objective function is 60, and after another local search, we are back to the same solution. Then, in Step 3, we take out 2 edges and add another 2 at random, and after a local search, an improved solution is obtained with a value of 39. Continuing in that fashion, we obtain in Step 8 the optimal solution with an objective function value equal to 36.

(iv) *General VNS.* Note that the local search Step 5 may also be replaced by VND (Algorithm 2). Using this general VNS (VNS/VND) approach led to some of the most successful applications reported (see, e.g., [1, 7, 9, 10, 25, 31, 52, 53]). The steps of general VNS (GVNS) are given in Algorithm 8 below.

Algorithm 8: Steps of the general VNS

Function GVNS $(x, \ell_{\max}, k_{\max}, t_{\max})$
repeat
  $k \leftarrow 1$
  repeat
    $x' \leftarrow$ Shake(x, k)
    $x'' \leftarrow$ VND$(x', \ell_{\max})$
    NeighborhoodChange(x, x'', k)
  until $k = k_{\max}$
  $t \leftarrow$ CpuTime()
until $t > t_{\max}$

(v) The *skewed VNS* (SVNS) method [23] addresses the problem of exploring valleys far from the incumbent solution. Indeed, once the best solution in a large region has been found, it is necessary to go quite far to obtain an improved one. Solutions drawn at random in faraway neighborhoods may differ substantially from the incumbent, and VNS may then degenerate, to some extent, into the multistart heuristic (in which descents are made iteratively from solutions generated at random, and that is known not to be very efficient). So some compensation for distance from the incumbent must be made, and a scheme called skewed VNS is proposed for that purpose. Its steps are presented in Algorithms 9, 10, and 11. The KeepBest(x, x') function in Algorithm 10 simply keeps the better one of solutions x and x'.

Algorithm 9: Steps of neighborhood change for the skewed VNS

Function NeighborhoodChangeS(x, x'', k, α)
if $f(x'') - \alpha\rho(x, x'') < f(x)$ **then**
  $x \leftarrow x''$; $k \leftarrow 1$
else
  $k \leftarrow k + 1$

SVNS makes use of a function $\rho(x, x'')$ to measure distance between the incumbent solution x and the local optimum found x''. The distance function used to define $\mathcal{N}_k$, as in the above examples, could be used also for this purpose. The parameter α must be chosen in such a way to accept movement to valleys faraway from x when $f(x'')$ is larger than $f(x)$ but not too much larger (otherwise one will

Algorithm 10: Steps of the skewed VNS

Function SVNS $(x, k_{\max}, t_{\max}, \alpha)$
1 **repeat**
2 $k \leftarrow 1$; $x_{\text{best}} \leftarrow x$
3 **repeat**
4 $x' \leftarrow$ Shake(x, k)
5 $x'' \leftarrow$ FirstImprovement(x')
6 KeepBest (x_{best}, x)
7 NeighborhoodChangeS(x, x'', k, α)
until $k = k_{\max}$
8 $x \leftarrow x_{\text{best}}$
9 $t \leftarrow$ CpuTime()
until $t > t_{\max}$

Algorithm 11: Keep the better solution

Function KeepBest(x, x')
1 **if** $f(x') < f(x)$ **then**
2 $x \leftarrow x'$

always leave x). A good value for α is to be found experimentally in each case. Moreover, in order to avoid frequent moves from x to a close solution, one may take a smaller value for α when $\rho(x, x'')$ is small. More sophisticated choices for a function of $\alpha\rho(x, x'')$ could be made through some learning process.

(vi) *Variable neighborhood decomposition search* (VNDS). The VNDS method [30] extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem. Its steps are presented in Algorithm 12, where t_d is an additional parameter and represents the running time given for solving decomposed (smaller sized) problems by VNS.

For ease of presentation, but without loss of generality, we assume that the solution x represents the set of some elements. In Step 4 we denote with y a set of k solution attributes present in x' but not in x ($y = x' \setminus x$). In Step 5 we find the local optimum y' in the space of y; then we denote with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$). We notice that exploiting some *boundary effects* in a new solution can significantly improve the solution quality. That is why, in Step 6, we find the local optimum x''' in the whole space S using x'' as an initial solution. If this is time-consuming, then at least a few local search iterations should be performed.

VNDS can be viewed as embedding the classical successive approximation scheme (which has been used in combinatorial optimization at least since the 1960s; see, e.g., [21]) in the VNS framework.

Algorithm 12: Steps of VNDS

Function VNDS $(x, k_{\max}, t_{\max}, t_d)$
repeat
  $k \leftarrow 1$
  repeat
    $x' \leftarrow$ Shake (x, k); $y \leftarrow x' \setminus x$
    $y' \leftarrow \text{VNS}(y, k, t_d)$; $x'' = (x' \setminus y) \cup y'$
    $x''' \leftarrow$ FirstImprovement(x'')
    NeighborhoodChange(x, x''', k)
  until $k = k_{\max}$
until $t > t_{\max}$

Some Extensions

(i) *Primal-dual VNS*. For most modern heuristics, the difference in value between the optimal solution and the obtained one is completely unknown. Guaranteed performance of the primal heuristic may be determined if a lower bound on the objective function value is known. To that end, the standard approach is to relax the integrality condition on the primal variables, based on a mathematical programming formulation of the problem. However, when the dimension of the problem is large, even the relaxed problem may be impossible to solve exactly by standard commercial solvers. Therefore, it seems a good idea to solve dual relaxed problems heuristically as well. In this way we get guaranteed bounds on the primal heuristic's performance. The next problem arises if we want to get an exact solution within a branch-and-bound framework since having the approximate value of the relaxed dual does not allow us to branch in an easy way, e.g., exploiting complementary slackness conditions. Thus, the exact value of the dual is necessary.

In primal-dual VNS (PD-VNS) [22] one possible general way to get both the guaranteed bounds and the exact solution is proposed. Its steps are given in Algorithm 13. In the first stage a heuristic procedure based on VNS is used to

Algorithm 13: Steps of the basic PD-VNS

Function PD-VNS $(x, \ell_{\max}, k_{\max}, t_{\max})$
BVNS $(x, \ell_{\max}, k_{\max}, t_{\max})$ // Solve primal by VNS
DualFeasible(x, y) // Find (infeasible) dual such that $f_P = f_D$
DualVNS(y) // Use VNS do decrease infeasibility
DualExact(y) // Find exact (relaxed) dual
BandB(x, y) // Apply branch-and-bound method

obtain a near-optimal solution. In [22] it is shown that VNS with decomposition is a very powerful technique for large-scale simple plant location problems (SPLP) with up to 15,000 facilities and 15,000 users. In the second phase the objective is to find an exact solution of the relaxed dual problem. Solving the relaxed dual is accomplished in three stages: (i) find an initial dual solution (generally infeasible) using the primal heuristic solution and complementary slackness conditions, (ii) find a feasible solution by applying VNS to the unconstrained nonlinear form of the dual, and (iii) solve the dual exactly starting with initial feasible solution using a customized "sliding simplex" algorithm that applies "windows" on the dual variables substantially reducing the size of the problem. In all problems tested, including instances much larger than previously reported in the literature, the procedure was able to find the exact dual solution in reasonable computing time. In the third and final phase armed with tight upper and lower bounds, obtained respectively, from the heuristic primal solution in phase one and the exact dual solution in phase two, we apply a standard branch-and-bound algorithm to find an optimal solution of the original problem. The lower bounds are updated with the dual sliding simplex method and the upper bounds whenever new integer solutions are obtained at the nodes of the branching tree. In this way it was possible to solve exactly problem instances of sizes up to 7000×7000 for uniform fixed costs and $15{,}000 \times 15{,}000$ otherwise.

(ii) *Variable neighborhood formulation space search*. Traditional ways to tackle an optimization problem consider a given formulation and search in some way through its feasible set X. The fact that a same problem may often be formulated in different ways allows to extend search paradigms to include jumps from one formulation to another. Each formulation should lend itself to some traditional search method, its "local search" that works totally within this formulation, and yields a final solution when started from some initial solution. Any solution found in one formulation should easily be translatable to its equivalent solution in any other formulation. We may then move from one formulation to another using the solution resulting from the former's local search as initial solution for the latter's local search. Such a strategy will of course only be useful when local searches in different formulations behave differently.

This idea was recently investigated in [43] using an approach that systematically changes formulations for solving circle packing problems (CPP). It is shown there that a stationary point of a nonlinear programming formulation of CPP in Cartesian coordinates is not necessarily also a stationary point in a polar coordinate system. A method called *reformulation descent* (RD) that alternates between these two formulations until the final solution is stationary with respect to both is suggested. Results obtained were comparable with the best known values, but they were achieved some 150 times faster than by an alternative single formulation approach. In the same paper the idea suggested above of *formulation space search* (FSS) is also introduced, using more than two formulations. Some research in that direction has been reported in [32, 43, 44, 50]. One methodology that uses variable neighborhood

idea in searching through the formulation space is given in Algorithms 14 and 15. Here ϕ (ϕ') denotes a formulation from given space $\mathcal{F}$, x (x') denotes a solution in the feasible set defined with that formulation, and $\ell \leq \ell_{\max}$ is the formulation neighborhood index. Note that Algorithm 15 uses a reduced VNS strategy in $\mathcal{F}$.

Algorithm 14: Formulation change function

Function `FormulationChange`$(x, x', \phi, \phi', \ell)$
if $f(\phi', x') < f(\phi, x)$ **then**
  $\phi \leftarrow \phi'$; $x \leftarrow x'$; $\ell \leftarrow \ell_{\min}$
else
  $\ell \leftarrow \ell + \ell_{\text{step}}$

Algorithm 15: Reduced variable neighborhood FSS

Function `VNFSS`$(x, \phi, \ell_{\max})$
repeat
  $\ell \leftarrow 1$ // Initialize formulation in $\mathcal{F}$
  while $\ell \leq \ell_{\max}$ **do**
    `ShakeFormulation`(x,x',ϕ,ϕ',ℓ) // $(\phi',x') \in (N_\ell(\phi),\mathcal{N}(x))$ at random
    `FormulationChange`(x,x',ϕ,ϕ',ℓ) // Change formulation
until *some stopping condition is met*

(iii) *Variable formulation search*. Many optimization problems in the literature, like min-max type of problems, present a flat landscape. This means that, given a formulation of the problem, there are many neighboring solutions with the same value of the objective function. When this happens, it is difficult to determine which neighborhood solution is more promising to continue the search. To address this drawback, the use of alternative formulations of the problem within VNS is proposed in [41, 46, 49]. In [49] it is named variable formulation search (VFS). It combines the change of neighborhood within the VNS framework, with the use of alternative formulations. In particular, the alternative formulations will be used to compare different solutions with the same value of the objective function, when considering the original formulation.

Let us assume that, besides the original formulation and the corresponding objective function $f_0(x)$, there are p other formulations denoted as $f_1(x), \ldots, f_p(x), x \in X$. Note that two formulations are equivalent if the optimal solution of one is the optimal solution of the other and vice versa. Without loss of clarity, we will denote different formulations as different objectives $f_i(x), i = 1, \ldots, p$. The idea of VFS is to add the procedure `Accept`(x, x', p), given in

Algorithm 16 in all three basic steps of BVNS: `Shaking`, `LocalSearch`, and `NeighbourhoodChange`. Clearly, if a better solution is not obtained by any formulation among the p preselected, the move is rejected. The next iteration in the loop of Algorithm 16 will take place only if the objective function values according to all previous formulations are equal.

Algorithm 16: Accept procedure with p secondary formulations

```
1: logical function Accept (x, x', p)
2: for i = 0 to p do
3:    condition1 = f_i(x') < f_i(x)
4:    condition2 = f_i(x') > f_i(x)
5:    if (condition1) then
6:       Accept← True; return
7:    else if (condition2) then
8:       Accept ← False return
9:    end if
10: end for
```

If `Accept` (x, x', p) is included into `LocalSearch` subroutine of BVNS, then it will not stop the first time a non-improved solution is found. In order to stop `LocalSearch` and thus claim that x' is local minimum, x' should not be improved by any among the p different formulations. Thus, for any particular problem, one needs to design different formulations of the problem considered and decide the order they will be used in the `Accept` subroutine. Answers to those two questions are problem specific and sometimes not easy. The `Accept` (x, x', p) subroutine can obviously be added to `NeighbourhoodChange` and `Shaking` steps of BVNS from Algorithm 7 as well.

In [46], three evaluation functions, or acceptance criteria, within `Neighborhood Change` step are used in solving **bandwith minimization problem**. This min-max problem consists in finding permutations of rows and columns of a given square matrix such that the maximal distance of nonzero element from the main diagonal in the corresponding row is minimum. Solution x may be presented as a labeling of a graph and the move from x to x' as $x \leftarrow x'$. Three criteria used are (1) the simplest one is based on the objective function value $f_0(x)$ (bandwidth lenth), (2) the total number of critical vertices $f_1(x)$ $(f_1(x') < f_1(x)|)$, and (3) $f_3(x, x') = \rho(x, x') - \alpha$, if $(f_0(x') = f_0(x)$ and $f_1(x') = f_1(x))$, but x and x' are relatively far one from another $\rho(x, x') > \alpha$, where α is an additional parameter. The idea for a move to even worse solution if it is very far is used within skewed VNS. However, move to the solution with the same value only if its Hamming distance from the incumbent is greater than α that is performed in [46].

In [41] different mathematical programming formulation of the original problem is used as a secondary objective within `Neighborhood Change` function of VNS. There, two combinatorial optimization problems on the graph are considered: **metric dimension problem** and **minimal doubly resolving set problem**.

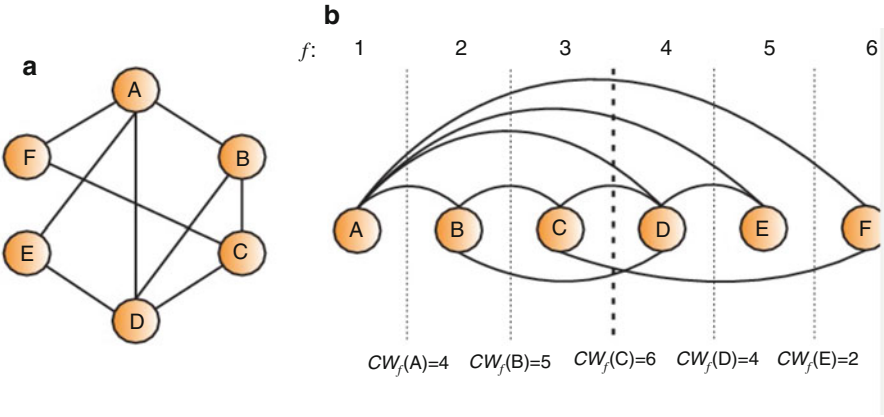

Fig. 4 Cutwidth minimization example as in [49]

Table 1 Impact of the use of alternative formulations in the search process, within 30 s

	BVNS	VFS_1	VFS_2	VFS_3
Avg.	137.31	93.56	91.56	90.75
Dev. (%)	192.44	60.40	49.23	48.22

More general VFS approach is done in [49], where the **cutwidth graph minimization problem** (CWP) is considered. CWP also belongs to min-max problem family. For a given graph, one needs to make sequence of nodes such that the maximum cutwidth is minimum. The definition of the cutwidth of the graph is clear from Fig. 4, where at (a) graph G with six vertices and nine edges is given.

At Fig. 4b, ordering x of the vertices of the graph in (a) with the corresponding cutwidth CW values of each vertex is presented. It is clear that the CW represents the number of cut edges between two consecutive nodes in the solution x. The cutwidth value $f_0(x) = CW(x)$ of the ordering $x = (A, B, C, D, E, F)$ is equal to $f_0(x) = \max\{4, 5, 6, 4, 2\} = 6$ (see Fig. 4). One needs to find order x that minimizes the maximum cutwidth value of each vertex.

Two additional formulations are used in [49] and implemented within VND local search. In Table 1 the results obtained with four different VFS variants, when executing them for 30 s over each instance of the *test* data set, are presented. The column BVNS represents a heuristic based on the BVNS which make use only of the original formulation of the CMP. VFS_1 denotes BVNS heuristic that uses only one secondary criterion. VFS_2 is equivalent to the previous one with the difference that now only f_2 is considered (instead of f_1). Finally, the fourth column of the table, denoted as VFS_3, combines the original formulation of the CMP with the two alternative ones, in the way presented in Algorithm 16. All the algorithms were configured with $k_{\max} = 0.1n$ and they start from the same random solution. It appears that the significant improvements in the solution quality are obtained when at least one secondary formulation is used in case of ties (compare, e.g., 192.44 % and 60.40 % deviations from the best known solutions reported by BVNS and VFS_1,

Table 2 Comparison with the state-of-the-art algorithms over the grid and HB data sets

	81 'grid' test instances				86 HB instances			
	GPR [2]	SA [12]	SS [48]	VFS [49]	GPR [2]	SA [12]	SS [48]	VFS [49]
Avg.	38.44	16.14	13.00	12.23	364.83	346.21	315.22	314.39
Dev. (%)	201.81	25.42	7.76	3.25	95.13	53.30	3.40	1.77
#Opt.	2	37	44	59	2	8	47	61
CPU t (s)	235.16	216.14	210.07	90.34	557.49	435.40	430.57	128.12

respectively). Additional improvement is obtained if all three formulations are used (in VFS_3).

Comparison of VFS_3 and state-of-the-art heuristics are given in Table 2. It appears that the best quality results are obtained by VFS in less computing time.

VNS for Mixed-Integer Linear Programming

The mixed-integer linear programming (MILP) problem consists of maximizing or minimizing a linear function, subject to equality or inequality constraints and integrality restrictions on some of the variables. The mixed integer programming problem (P) can be expressed as

$$(MILP) \quad \left[\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in M = \{1, 2, \ldots, m\} \\ & x_j \in \{0, 1\} & \forall j \in \mathcal{B} \neq \emptyset \\ & x_j \geq 0, \texttt{integer} & \forall j \in \mathcal{G} \\ & x_j \geq 0 & \forall j \in \mathcal{C} \end{array}\right.$$

where the set of indices $N = \{1, 2, \ldots, n\}$ is partitioned into three subsets $\mathcal{B}, \mathcal{G}$, and $\mathcal{C}$, corresponding to binary, general integer, and continuous variables, respectively.

Numerous combinatorial optimization problems, including a wide range of practical problems in business, engineering, and science, can be modeled as MILP problems. Its several special cases, such as knapsack, set packing, cutting and packing, network design, protein alignment, traveling salesman, and other routing problems, are known to be NP-hard [19]. There are several commercial solvers such as CPLEX [33] for solving MILPs. Methods included in such software packages are usually of branch-and-bound (B&B) or of branch-and-cut (B&C) types. Basically, those methods enumerate all possible integer values in some order and perform some restrictions for the cases where such enumeration cannot improve the currently best solution.

The connection between local search type of heuristics and exact solvers may be established by introducing the so-called local branching constraint [17]. By adding just one constraint into the (MILP), the kth neighborhood of MILP is defined.

This allows the use of all local search-based Metaheuristics, such as tabu search, simulating annealing, VNS, etc. More precisely, given two solutions x and y of the problem (MILP), we define the distance between x and y as

$$\delta(x, y) = \sum_{j \in \mathcal{B}} \mid x_j - y_j \mid.$$

Let X be the solution space of the problem (MILP) considered. The neighborhood structures $\{\mathcal{N}_k \mid k = 1, \ldots, k_{\max}\}$ can be defined, knowing the distance $\delta(x, y)$ between any two solutions $x, y \in X$. The set of all solutions in the kth neighborhood of $y \in X$ is denoted as $\mathcal{N}_k(y)$ where

$$\mathcal{N}_k(y) = \{x \in X \mid \delta(x, y) \leq k\}.$$

For the pure 0-1 MILP given above, ($\mathcal{G} = \emptyset$, $\delta(.,.)$ represents the Hamming distance, and $\mathcal{N}_k(y)$ may be expressed by the following so-called local branching constraint:

$$\delta(x, y) = \sum_{j \in S} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k, \tag{2}$$

where $S = \{j \in \mathcal{B} \mid y_j = 1\}$.

In [31] a general VNS procedure for solving 0-1 MILPs is developed (see Algorithm 17).

An exact MILP solver (CPLEX) is used as a black box for finding the best solution in the neighborhood, based on the given formulation (MILP) plus the added local branching constraints. Shaking is performed using the Hamming distance defined above. The detailed explanation of VNB method given in Algorithm 17 below and the meaning of all variables (*x_cur, x_opt, UB, first, f_opt, cont, rhs,* etc.) and constants (`total_time_limit, node_time_limit, k_step`) used may be found in [31].

VNS for Continuous Global Optimization

The continuous constrained nonlinear global optimization problem (GOP) in general form is given as follows:

$$\text{(GOP)} \quad \begin{bmatrix} \min & f(x) & \\ \text{s.t.} & g_i(x) \leq 0 & \forall i \in \{1, 2, \ldots, m\} \\ & h_i(x) = 0 & \forall i \in \{1, 2, \ldots, r\} \\ & a_j \leq x_j \leq b_j & \forall j \in \{1, 2, \ldots, n\} \end{bmatrix}$$

where $x \in R^n$, $f : R^n \to R$, $g_i : R^n \to R$, $i = 1, 2, \ldots, m$, and $h_i : R^n \to R$, $i = 1, 2, \ldots, r$ are possibly nonlinear continuous functions, and $a, b \in R^n$ are the variable bounds.

Algorithm 17: Steps of the VNS branching

```
Function VnsBra(total_time_limit, node_time_limit, k_step, x_opt)
 1  TL := total_time_limit; UB := ∞; first := true
 2  stat := MIPSOLVE(TL, UB, first, x_opt, f_opt)
 3  x_cur:=x_opt; f_cur:=f_opt
 4  while (elapsedtime < total_time_limit) do
 5      cont := true; rhs := 1; first := false
 6      while (cont or elapsedtime < total_time_limit) do
 7          TL = min(node_time_limit, total_time_limit-elapsedtime)
 8          add local br. constr. δ(x, x_cur) ≤ rhs; UB := f_cur
 9          stat := MIPSOLVE(TL, UB, first, x_next, f_next)
10          switch stat do
11              case "opt_sol_found":
12                  reverse last local br. const. into δ(x, x_cur) ≥ rhs + 1
13                  x_cur := x_next; f_cur := f_next; rhs := 1;
14              case "feasible_sol_found":
15                  reverse last local br. constr. into δ(x, x_cur) ≥ 1
16                  x_cur := x_next; f_cur := f_next; rhs := 1;
17              case "proven_infeasible":
18                  remove last local br. constr.; rhs := rhs+1;
19              case "no_feasible_sol_found":
20                  cont := false
21      if f_cur < f_opt then
22          x_opt := x_cur; f_opt := f_cur; k_cur := k_step;
        else
23          k_cur := k_cur+k_step;
24      remove all added constraints; cont := true
25      while cont and (elapsedtime < total_time_limit) do
26          add constraints k_cur ≤ δ(x, x_opt) < k_cur + k_step
27          TL := total_time_limit-elapsedtime; UB := ∞; first := true
28          stat := MIPSOLVE(TL, UB, first, x_cur, f_cur)
29          remove last two added constraints; cont =false
30          if stat = "proven_infeasible" or "no_feasible" then
31              cont :=true; k_cur := k_cur+k_step
```

GOP naturally arises in many applications, e.g., in advanced engineering design, data analysis, financial planning, risk management, scientific modeling, etc. Most cases of practical interest are characterized by multiple local optima, and, therefore, in order to find the globally optimal solution, a global scope search effort is needed.

If the feasible set X is convex and objective function f is convex, then GOP is relatively easy to solve, i.e., the Karush-Kuhn-Tucker conditions may be applied. However, if X is not a convex set or f is not a convex function, we could have many local minima, and thus, the problem may not be solved by using classical techniques.

For solving GOP, VNS has been used in two different ways: (a) with neighborhoods induced by using an ℓ_p norm and (b) without using an ℓ_p norm.

(a) VNS with ℓ_p norm neighborhoods [3, 4, 15, 36, 39, 42]. A natural approach in applying VNS for solving GOP is to induce neighborhood structures $\mathcal{N}_k(x)$ from the ℓ_p metric:

$$\rho(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty) \tag{3}$$

or

$$\rho(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p \to \infty). \tag{4}$$

The neighborhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k-th neighborhood of x, and using the metric ρ, it is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x, y) \leq \rho_k\}, \tag{5}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} \leq \rho(x, y) \leq \rho_k\}, \tag{6}$$

where ρ_k, known as the radius of $\mathcal{N}_k(x)$, is monotonically increasing with k.

For solving box constraint GOP, both [36] and [15] use neighborhoods as defined in (6). The basic differences between the two are as follows: (1) in the procedure suggested in [36] the ℓ_∞ norm is used, while in [15] the choice of metric is either left to the analyst or changed automatically according to some predefined order; (2) as a local search procedure within VNS, the commercial solver SNOPT [20] is used in [36], while in [15] the analyst may choose one out of six different convex minimizers. A VNS-based heuristic for solving the generally constrained GOP is suggested in [39]. There, the problem is first transformed into a sequence of box-constrained problems within well-known exterior point method:

$$\min_{a \leq x \leq b} F_{\mu,q}(x) = f(x) + \frac{1}{\mu} \sum_{i=1}^{m} (\max\{0, q_i(x)\})^q + \sum_{i=1}^{r} |h_i(x)|^q, \tag{7}$$

where μ and $q \geq 1$ are a positive penalty parameter and penalty exponent, respectively. Algorithm 18 outlines the steps for solving the box constraint subproblem as proposed in [39]:

The `Glob-VNS` procedure from Algorithm 18 contains the following parameters in addition to $k_{\max}$ and $t_{\max}$:

1. *Values of radii* ρ_k, $k = 1, \ldots, k_{\max}$. Those values may be defined by the user or calculated automatically in the minimizing process;

Algorithm 18: Steps of continuous VNS using ℓ_p norm

Function `Glob-VNS` $(x^*, k_{\max}, t_{\max})$
1 Select the set of neighborhood structures $\mathcal{N}_k$ $k = 1, \ldots, k_{\max}$
2 Select the array of random distributions types and an initial point $x* \in X$
3 $x \leftarrow x^*$, $f^* \leftarrow f(x)$, $t \leftarrow 0$
4 **while** $t < t_{\max}$ **do**
5 $\quad k \leftarrow 1$
6 $\quad$ **repeat**
7 $\quad\quad$ **for** *all distribution types* **do**
8 $\quad\quad\quad y \leftarrow$ `Shake`(x^*, k) // Get $y \in \mathcal{N}_k(x^*)$ at random
9 $\quad\quad\quad y' \leftarrow$ `BestImprovment`(y) // Apply LS to obtain a local minimum y'
10 $\quad\quad\quad$ **if** $f(y') < f^*$ **then**
11 $\quad\quad\quad\quad x^* \leftarrow y'$, $f^* \leftarrow f(y')$, `go to line 5`
12 $\quad\quad k \leftarrow k + 1$
$\quad$ **until** $k = k_{\max}$
13 $\quad t \leftarrow$ `CpuTime()`

2. *Geometry* of neighborhood structures $\mathcal{N}_k$, defined by the choice of metric. Usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms;
3. *Distribution* used for obtaining the random point y from $\mathcal{N}_k$ in the *shaking* step. Uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to much better performance on some problems.

Different choices of geometric neighborhood shapes and random point distributions lead to different VNS-based heuristics.

(b) VNS without using ℓ_p norm. Two different neighborhoods, $N_1(x)$ and $N_2(x)$, are used in VNS-based heuristic suggested in [55]. In $N_1(x)$, r (a parameter) random directions from the current point x are generated and one-dimensional searches along each performed. The best point (out of r) is selected as a new starting solution for the next iteration, if it is better than the current one. If not, as in VND, the search is continued within the next neighborhood $N_2(x)$. The new point in $N_2(x)$ is obtained as follows. The current solution is moved parallel to each x_j $(j = 1, \ldots, n)$ by value Δ_j, taken at random from interval $(-\alpha, \alpha)$, i.e., $x_j^{(\text{new})} = x_j + \Delta_j$ or $x_j^{(\text{new})} = x_j - \Delta_j$. Points obtained by the plus or minus sign for each variable define neighborhood $N_2(x)$. If change of $x_j^{(\text{new})}$ by 1 % to the right gives a better solution than in $x^{(\text{new})}$, the + sign is chosen; otherwise the – sign is chosen.

Neighborhoods N_1 and N_2 are used for designing two algorithms. The first, called VND, iterates these neighborhoods until there is no improvement in the

solution value. In the second variant, a local search is performed in N_2 and $k_{\max}$ set to 2 for the shaking step. In other words, a point from neighborhood $k = 2$ is obtained by generating a random direction followed by line search along it (as prescribed for N_1) and then by changing each of the variables (as prescribed for N_2).

Variable Neighborhood Programming

Building an intelligent machine is an old dream that, thanks to computers, began to take shape. Automatic programming is an efficient technique that has contributed an important development in the field of artificial intelligence. Genetic programming (GP) [35], inspired by genetic algorithm (GA), is among the few evolutionary algorithms used to evolve population of programs. The main difference between GP and GA is the presentation of a solution. An individual in GA can be a string, while GPs individuals are programs. The usual way to present program within GP is by using a tree. For example, assume that the current solution of the problem is the following function:

$$f(x_1, \ldots, x_5) = \frac{x_1}{x_2 + x_3} + x_4 - x_5.$$

Then the code (tree) that calculates it using GP may be presented as in Fig. 5a.

Souhir et al. [16] recently adapted VNS rules in solving automatic programming problem. They first suggested an extended solution presentation by adding coefficients to variables. Each terminal node was attached to its own parameter value. These parameters gave a weight for each terminal node, with values from the interval [0, 1]. This form of presentation allowed VNP to examine parameter values and the

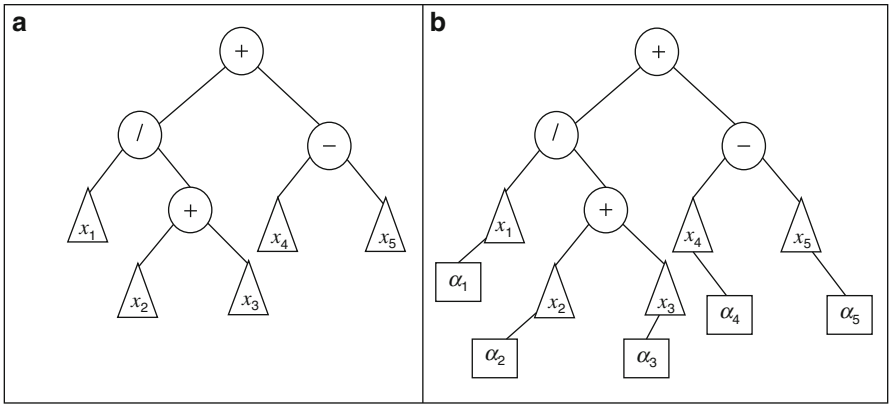

Fig. 5 Current solution representation in automatic programming problem: $\frac{x_1}{x_2+x_3} + x_4 - x_5$. (**a**) GP solution representation. (**b**) VNP solution representation

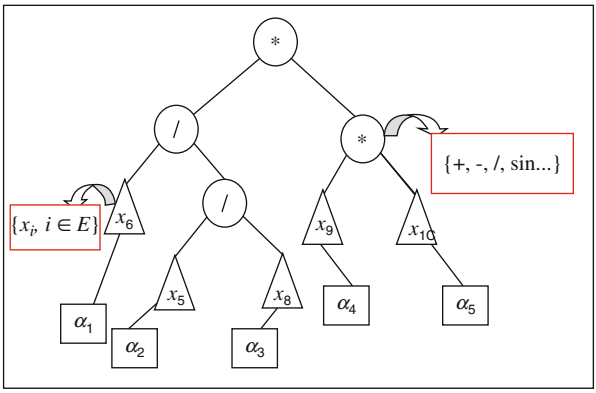

Fig. 6 Neighborhood structure N_1: changing a node value

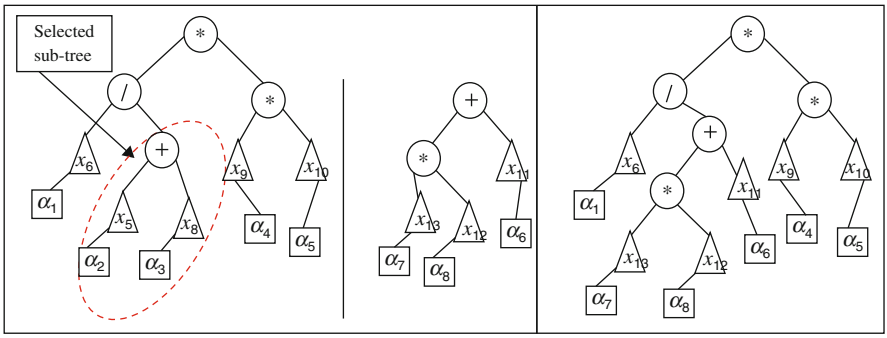

Fig. 7 Neighborhood structure N_2: swap operator

remaining tree structures in the same iteration. Let $G = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ denote a parameter set. In Fig. 5b an example of VNP's solution representation is illustrated.

Neighborhood structures. At such solution presentation as a tree T, nine different neighborhood structures are proposed in [16]. To save the space, we will just mention some of them:

- $N_1(T)$ – **Changing a node value operator.** This neighborhood structure conserves the skeleton of the tree and changes only the values of a functional or a terminal node. Each node can obtain many values from its corresponding set. Let x_i be the current solution; its neighbor x_{i+1} differs from x_i by just a single node. A move within this neighborhood structure is shown in Fig. 6.
- $N_2(T)$ – **Swap operator**. By this operator a first node from the current tree is taken at random, and a new sub-tree is generated as presented in Fig. 7a, b. Then the selected node is attached in the place of the sub-tree. In this move, the constraint related to the maximum tree size should be respected. More details can be seen in Fig. 7.

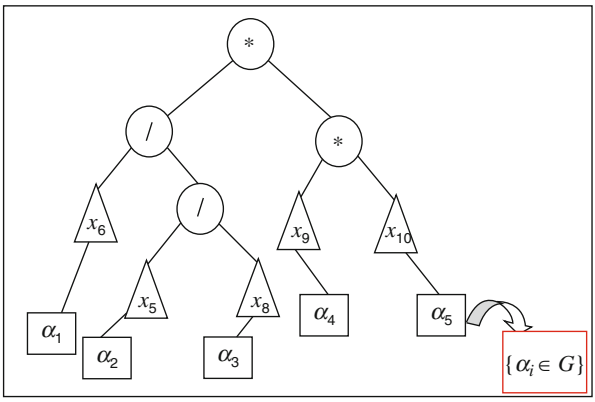

Fig. 8 Neighborhood structure N_3: change parameters

- $N_3(T)$ – **Changing parameter values.** In the previous neighborhood structures, the tree form and its node values were considered. In $N_3(T)$ neighborhood, attention is paid on parameters. So, the position and value of nodes are kept in order to search the neighbors in the parametric space. Figure 8 illustrates details. The change from one value to another is at random.

These neighborhood structures may be used in both local search step ($N_\ell, \ell \in [1, \ell_{\max}]$) and in the shaking step ($\mathcal{N}_k, k \in [1, k_{\max}]$) of the VNS.

VNP shaking. The shaking step allows the diversification in the search space. Our proposed VNP algorithm does not use exactly the same neighborhood structures N_ℓ as for the local search. That is the reason why neighborhoods used in shaking are denoted differently: $\mathcal{N}_k(T), k = 1, k_{\max}$. $\mathcal{N}_k(T)$ may be constructed by repeating k times one or more moves from the set $\{N_\ell(T), |\ell = 1, \ldots, \ell_{\max}\}$, explained earlier. Nevertheless, in the shaking phase, we use a neighborhood structure that mainly affects the skeleton of a presented solution with its different forms for the perturbation. For more clear explanation, we take the swap operator $N_3(T)$ as an example. Let m denote the maximum size of a solution. We can get a solution from the kth neighborhood of T using the swap operator: k may represent the size of the new generated sub-tree. If n denotes the size of the original tree after deleting the old sub-tree, then $n + k_{\max} \leq m$. The objective of this phase is to provide a good starting point for the local search.

VNP objective function. The evaluation consists of defining a fitness or objective function assessing the proposed problem. This function is defined according to the problem considered. After running each solution (program) on training data set, fitness may be measured by counting how many training cases the current solution result is correct or near to the exact solution.

An example: time series forecasting (TSF) problem. Two widely used benchmark data sets of TSF problem are explored in [16] to examine VNP capabilities: Mackey-Glass series and Box-Jenkins set. The parameters for the VNP implementation are chosen after some preliminary testing are given in Table 3.

Table 3 VNP parameters adjustment for the forecasting problem

Parameters	Values
The functional set	$F = \{+, *, , pow\}$;
The terminal sets	$\{(x_i, c), i \in [1, \ldots, m], m =$ number of inputs, $c \in R\}$;
Neighborhood structures	$\{N_1, N_2, N_3\}$;
Minimum tree length	20 nodes;
Maximum tree length	200 nodes;
Maximum number of iterations	50,000

Table 4 Comparison of testing error on Box-Jenkins dataset

Method	Prediction error RMSE
ODE [54]	0.5132
HHMDDE [14]	0.3745
FBBFNT [6]	0.0047
VNP [16]	**0.0038**

The root-mean-square error (RMSE) is used as fitness function, as it is usual in the literature:

$$f(T) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_t^j - y_{\text{out}}^j)^2}$$

where n represents the total number of samples and y_{out}^j and y_t^j are outputs of the sample number j obtained by the VNP model and the desired one, respectively. Here we will just illustrate comparison on Box-Jenkins instance.

The gas furnace data of Box and Jenkins were collected from a combustion process of a methane-air mixture [15]. This time series has found a widespread application as a benchmark example in many practical sciences for testing prediction algorithms. The data set contains 296 pairs of input-output values. The input $u(t)$ corresponds to the gas flow, and the output $y(t)$ presents the CO_2 concentration in outlet gas. The inputs are $u(t-4)$ and $y(t-1)$, and the output is $y(t)$. In this work, 200 samples are used in the training phase, and the remaining samples are used for the testing phase. The performance of the evolved model is evaluated by comparing it with the abovementioned approaches. The RMSE achieved by VNP output model is 0.0038, which appeared to be better than the RMSE obtained by other approaches. Table 4 shows that VNP approach proves effectiveness and generalization ability.

Discovery Science

In all the above applications, VNS is used as an optimization tool. It can also lead to results in "discovery science," i.e., help in the development of theories. This has been done for graph theory in a long series of papers with the common title "Variable

neighborhood search for extremal graphs" and reporting on the development and applications of the system AutoGraphiX (AGX) [10]. This system addresses the following problems:

- Find a graph satisfying given constraints;
- Find optimal or near optimal graphs for an invariant subject to constraints;
- Refute a conjecture;
- Suggest a conjecture (or repair or sharpen one);
- Provide a proof (in simple cases) or suggest an idea of proof.

Then a basic idea is to consider all of these problems as parametric combinatorial optimization problems on the infinite set of all graphs (or in practice some smaller subset) with a generic heuristic. This is done by applying VNS to find extremal graphs, with a given number n of vertices (and possibly also a given number of edges). Then a VND with many neighborhoods is used. Those neighborhoods are defined by modifications of the graphs such as the removal or addition of an edge, rotation of an edge, and so forth. Recently all moves involving four vertices or less were considered jointly in learning optimization framework. Once a set of extremal or near-extremal graphs, parameterized by their order, is found, their properties are explored with various data mining techniques, leading to conjectures, refutations, and simple proofs or ideas of proof.

All papers in this area are divided into the following groups in [28], where the extensive list of references for each class of problems can be found:

(i) *Principles of the approach* and its implementation;
(ii) *Applications to spectral graph theory*, e.g., finding bounds on the index or spectral radius of the adjacency matrix for various families of graphs and finding graphs maximizing or minimizing the index subject to some conditions;
(iii) *Studies of classical graph parameters*, e.g., independence, chromatic number, clique number, and average distance;
(iv) *Studies of little known or new parameters of graphs*, e.g., irregularity, proximity, and remoteness;
(v) *New families of graphs discovered by AGX*, e.g., bags, which are obtained from complete graphs by replacing an edge by a path, and bugs, which are obtained by cutting the paths of a bag;
(vi) *Applications to mathematical chemistry*, e.g., study of chemical graph energy and of the Randić index;
(vii) *Results of a systematic study of pairwise comparison of 20 graph invariants*, involving the four usual operators: plus, minus, ratio, and product, which led to almost 1500 new conjectures, more than half of which were automatically proved by AGX and over 300 by various mathematicians;
(viii) *Refutation or strengthening of conjectures from the literature*;
(ix) *Surveys and discussions about various discovery systems in graph theory*, assessment of the state-of-the-art and the forms of interesting conjectures together with proposals for the design of more powerful systems.

Conclusions

The general schemes of variable neighborhood search have been presented, discussed, and illustrated by examples. In order to evaluate the VNS research program, one needs a list of the desirable properties of Metaheuristics. The following eight of these are presented in Hansen and Mladenović [27]:

(i) *Simplicity:* the metaheuristic should be based on a simple and clear principle, which should be widely applicable;
(ii) *Precision:* the steps of the metaheuristic should be formulated in precise mathematical terms, independent of possible physical or biological analogies which may have been the initial source of inspiration;
(iii) *Coherence:* all steps of the heuristics for particular problems should follow naturally from the principle of the metaheuristic;
(iv) *Efficiency:* heuristics for particular problems should provide optimal or near-optimal solutions for all or at least most realistic instances. Preferably, they should find optimal solutions for most problems of benchmarks for which such solutions are known, when available;
(v) *Effectiveness:* heuristics for particular problems should take a moderate computing time to provide optimal or near-optimal solutions;
(vi) *Robustness:* the performance of heuristics should be consistent over a variety of instances, i.e., not merely fine-tuned to some training set and less good elsewhere;
(vii) *User-friendliness:* heuristics should be clearly expressed, easy to understand, and, most important, easy to use. This implies they should have as few parameters as possible, ideally none;
(viii) *Innovation:* preferably, the principle of the metaheuristic and/or the efficiency and effectiveness of the heuristics derived from it should lead to new types of application.
(ix) *Generality:* the metaheuristic should lead to good results for a wide variety of problems;
(x) *Interactivity:* the metaheuristic should allow the user to incorporate his knowledge to improve the resolution process;
(xi) *Multiplicity:* the metaheuristic should be able to present several near-optimal solutions from which the user can choose one.

As shown above, VNS possesses, to a great extent, all of the above properties. This has led to heuristics which are among the very best ones for many problems. Interest in VNS is clearly growing at speed. This is evidenced by the increasing number of papers published each year on this topic (15 years ago, only a few; 10 years ago, about a dozen; about 50 in 2007). According to Google Scholar, the first two papers on VNS were cited almost 4,000 times!

Moreover, the 18th and 28th EURO Mini Conferences were entirely devoted to VNS. It led to special issues of *IMA Journal of Management Mathematics* in 2007

[37], *European Journal of Operational Research* [29], *Journal of heuristics* [47] in 2008, *Computers and Operations Research* in 2014 [45], and *Journal of Global Optimization* in 2016 [11]. The 3rd International VNS meeting took place in Tunisia, and three new special issues entirely devoted to VNS are in preparation (*Computers and Operations Research*, *Optimization Letters*, and *Yugoslav Journal of Operations Research*). In retrospect, it appears that the good shape of the VNS research program is due to the following decisions, strongly influenced by Karl Popper's philosophy of science [51]: (i) in devising heuristics favor insight over efficiency (which comes later) and (ii) learn from the heuristics mistakes.

References

1. Aloise DJ, Aloise D, Rocha CTM, Ribeiro CC, Ribeiro JC, Moura LSS (2006) Scheduling workover rigs for onshore oil production. Discret Appl Math 154(5):695–702
2. Andrade DV, Resende MGC (2007) GRASP with path-relinking for network migration scheduling. In: Proceedings of international network optimization conference (INOC), Spa
3. Audet C, Báchard V, Le Digabel S (2008) Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search J Glob Optim 41(2):299–318
4. Audet C, Brimberg J, Hansen P, Mladenović N (2004) Pooling problem: alternate formulation and solution methods, Manag Sci 50:761–776
5. Belacel N, Hansen P, Mladenović N (2002) Fuzzy J-means: a new heuristic for fuzzy clustering. Pattern Recognit 35(10):2193–2200
6. Bouaziz S, Dhahri H, Alimi AM, Abraham A (2013) A hybrid learning algorithm for evolving flexible beta basis function neural tree model. Neurocomputing 117: 107–117. doi:10.1016/j.neucom.2013.01.024
7. Brimberg J, Hansen P, Mladenović N, Taillard É (2000) Improvements and comparison of heuristics for solving the multisource Weber problem. Oper Res 48(3):444–460
8. Brimberg J, Mladenović N (1996) A variable neighborhood algorithm for solving the continuous location-allocation problem. Stud Locat Anal 10:1–12
9. Canuto S, Resende M, Ribeiro C (2001) Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks 31(3):201–206
10. Caporossi G, Hansen P (2000) Variable neighborhood search for extremal graphs 1. The AutoGraphiX system. Discret Math 212:29–44
11. Carrizosa E, Hansen P, Moreno-Perez JA (2015). Variable neighborhood search. J Glob Optim (Spec Issue) 63(3):427–629
12. Cohoon J, Sahni S (1987) heuristics for backplane ordering. J VLSI Comput Syst 2:37–61
13. Davidon WC (1959) Variable metric algorithm for minimization. Argonne National Laboratory report ANL-5990
14. Dhahri H, Alimi AM, Abraham A (2012) Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network. Neurocomputing 97:131–140
15. Dražić M, Kovacevic-Vujcić V, Cangalović M, Mladenović N (2006) GLOB – a new VNS-based software for global optimization In: Liberti L, Maculan N (eds) Global optimization: from theory to implementation. Springer, New York, pp 135–144
16. Elleucha S, Jarbouia B, Mladenovic N (2015) Variable neighborhood programming a new automatic programming method in artificial intelligence, Gerad Technical report G-2016-21, HEC Montreal, Canada
17. Fischetti M, Lodi A (2003) Local branching. Math Program 98(1–3):23–47
18. Fletcher R, Powell MJD (1963) Rapidly convergent descent method for minimization. Comput J 6:163–168

19. Garey MR, Johnson DS (1978) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New-York
20. Gill P, Murray W, Saunders MA (2002) SNOPT: an SQP algorithms for largescale constrained optimization. SIAM J Optim 12(4):979–1006
21. Griffith RE, Stewart RA (1961) A nonlinear programming technique for the optimization of continuous processing systems. Manag Sci 7:379–392
22. Hansen P, Brimberg J, Urošević D, Mladenović N (2007) Primal-dual variable neighborhood search for the simple plant location problem. INFORMS J Comput 19(4):552–564
23. Hansen P, Jaumard B, Mladenović N, Parreira A (2000) Variable neighborhood search for weighted maximum satisfiability problem. Les Cahiers du GERAD G–2000–62, HEC Montréal
24. Hansen P, Mladenović N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130:449–467
25. Hansen P, Mladenović N (2001) J-means: a new local search heuristic for minimum sum-of-squares clustering. Pattern Recognit 34:405–413
26. Hansen P, Mladenović N (2001) Developments of variable neighborhood search. In: Ribeiro C, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer, Dordrecht/London, pp 415–440
27. Hansen P, Mladenović N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 145–184
28. Hansen P, Mladenović N, Brimberg J, Moreno-Perrez JA (2010) Variable neighborhood search. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, 2nd edn. Kluwer, New York, pp 61–86
29. Hansen P, Mladenović N, Moreno Pérez JA (2008) Variable neighborhood search. Eur J Oper Res 191(3):593–595
30. Hansen P, Mladenović N, Pérez-Brito D (2001) Variable neighborhood decomposition search. J Heuristics 7(4):335–350
31. Hansen P, Mladenović N, Urošević D (2006) Variable neighborhood search and local branching. Comput Oper Res 33(10):3034–3045
32. Hertz A, Plumettaz M, Zufferey N (2008) Variable space search for graph coloring. Discret Appl Math 156(13):2551–2560
33. ILOG (2006) CPLEX 10.1. User's manual
34. Jornsten K, Lokketangen A (1997) Tabu search for weighted k-cardinality trees. Asia-Pac J Oper Res 14(2):9–26
35. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
36. Liberti L, Dražić M (2005) Variable neighbourhood search for the global optimization of constrained NLPs. In: Proceedings of GO workshop, Almeria
37. Melián B, Mladenović N (2007) Editorial IMA J Manag Math 18(2):99–100
38. Mladenovic N (1995) Variable neighborhood algorithm – a new metaheuristic for combinatorial optimization. In: Optimization days conference, Montreal, p 112
39. Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization Eur J Oper Res 191(3):753–770
40. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24: 1097–1100
41. Mladenovic N, Kratica J, Kovacevic-Vujcic V, Cangalovic M (2012) Variable neighborhood search for metric dimension and minimal doubly resolving set problems. Eur J Oper Res 220(2):328–337
42. Mladenović N, Petrović J, Kovačević-Vujčić V, Čangalović M (2003) Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. Eur J Oper Res 151:389–399
43. Mladenović N, Plastria F, Urošević D (2005) Reformulation descent applied to circle packing problems. Comput Oper Res 32:2419–2434
44. Mladenović N, Plastria F, Uroševic D (2007) Formulation space search for circle packing problems. Engineering Stochastic local search algorithms. Designing, implementing and

analyzing effective heuristics. Lecture notes in computer science, vol 4638, pp 212–216. https://link.springer.com/book/10.1007/978-3-540-74446-7
45. Mladenovic N, Salhi S, Hnafi S, Brimberg J (eds) (2014) Recent advances in variable neighborhood search. Comput Oper Res 52(B):147–148
46. Mladenovic N, Urosevic D, Pérez-Brito D, García-González CG (2010) Variable neighbourhood search for bandwidth reduction. Eur J Oper Res 200(1):14–27
47. Moreno-Vega JM, Melián B (2008) Introduction to the special issue on variable neighborhood search. J Heuristics 14(5):403–404
48. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199:285–304
49. Pardo EG, Mladenovic N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252 (2013)
50. Plastria F, Mladenović N, Urošević D (2005) Variable neighborhood formulation space search for circle packing. In: 18th mini Euro conference VNS, Tenerife
51. Popper K (1959) The logic of scientific discovery Hutchinson, London
52. Ribeiro CC, de Souza MC (2002) Variable neighborhood search for the degree-constrained minimum spanning tree problem. Discret Appl Math 118(1–2):43–54
53. Ribeiro CC, Uchoa E, Werneck R (2002) A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J Comput 14(3):228–246
54. Subudhi B, Jena D (2011) A differential evolution based neural network approach to nonlinear system identification, Appl Soft Comput 11:861–871. doi:10.1016/j.asoc.2010.01.006
55. Toksari AD, Güner E (2007) Solving the unconstrained optimization problem by a variable neighborhood search. J Math Anal Appl 328(2):1178–1187
56. Whitaker R (1983) A fast algorithm for the greedy interchange of large-scale clustering and median location problems INFOR 21:95–108